**Exercise 1: Understanding Thread States**

**Objectives:**

* Learn about different states of a thread.
* Understand how threads transition between states.

**Business Scenario:**

You are developing a logging system that needs to monitor the states of various threads in a multithreaded application. You need to create a thread and observe its state transitions.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **ThreadStateLogger**.
2. **Implement Thread States Logging:**
   * Define a new thread class that overrides the **run** method to perform a simple task (e.g., printing numbers).
   * In the **ThreadStateLogger** class, create an instance of this thread.
   * Log the state of the thread at various points: before starting, after starting, during execution, and after completion.
3. **Execute the Program:**
   * Run the **ThreadStateLogger** class and observe the output showing the state transitions of the thread.

**Exercise 2: Creating and Running Threads**

**Objectives:**

* Create and start threads in Java.
* Understand the main thread and how other threads interact with it.

**Business Scenario:**

You are developing a simulation where multiple sensors collect data simultaneously. Each sensor should run in its own thread.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **SensorSimulation**.
2. **Define Sensor Threads:**
   * Create a thread class named **Sensor** that simulates data collection by printing random data at regular intervals.
   * In the **SensorSimulation** class, create and start multiple **Sensor** threads.
3. **Main Thread Interaction:**
   * In the **SensorSimulation** class, ensure the main thread waits for all sensor threads to complete before exiting.
   * Use **join** method to make the main thread wait for the sensor threads.
4. **Execute the Program:**
   * Run the **SensorSimulation** class and observe the concurrent execution of sensor threads.

**Exercise 3: Using Sleep, Yield, and Join**

**Objectives:**

* Understand the use of **sleep**, **yield**, and **join** methods in thread management.

**Business Scenario:**

You are developing a task scheduler where tasks need to be paused, yielded to other tasks, or waited upon.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **TaskScheduler**.
2. **Define Task Threads:**
   * Create a thread class named **Task** that performs a sequence of operations.
   * Use **sleep** to pause the task for a few seconds.
   * Use **yield** to allow other tasks to execute.
   * Use **join** to wait for a dependent task to complete.
3. **Implement Task Scheduling:**
   * In the **TaskScheduler** class, create and start multiple Task threads.
   * Demonstrate the use of **sleep**, **yield**, and **join** in the task threads.
4. **Execute the Program:**
   * Run the **TaskScheduler** class and observe the task scheduling behavior.

**Exercise 4: Synchronization**

**Objectives:**

* Understand synchronization in multithreaded applications.
* Avoid data inconsistency with synchronized methods and blocks.

**Business Scenario:**

You are developing a banking system where multiple threads perform transactions on the same account. Ensure that transactions are synchronized to avoid data inconsistency.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **BankAccount**.
2. **Define Account Operations:**
   * Create methods for deposit and withdrawal operations.
   * Use synchronized methods or blocks to ensure thread safety.
3. **Create Transaction Threads:**
   * Create a thread class named **Transaction** that performs deposit and withdrawal operations on a shared **BankAccount** instance.
4. **Implement and Test Synchronization:**
   * In the **BankAccount** class, create and start multiple **Transaction** threads.
   * Observe and ensure that the balance updates correctly with synchronized operations.
5. **Execute the Program:**
   * Run the **BankAccount** class and test the synchronized transaction operations.

**Exercise 5: Inter-Thread Communication**

**Objectives:**

* Learn about inter-thread communication using **wait**, **notify**, and **notifyAll**.

**Business Scenario:**

You are developing a producer-consumer system where multiple producer threads generate data and multiple consumer threads process data. Implement inter-thread communication to synchronize data production and consumption.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **ProducerConsumer**.
2. **Define Shared Resource:**
   * Create a class named **DataQueue** that holds the data queue and methods for adding and removing data.
   * Use **wait**, **notify**, and **notifyAll** methods for inter-thread communication.
3. **Create Producer and Consumer Threads:**
   * Create thread classes named **Producer** and **Consumer** that interact with the **DataQueue** to produce and consume data.
4. **Implement and Test Inter-Thread Communication:**
   * In the **ProducerConsumer** class, create and start multiple **Producer** and **Consumer** threads.
   * Ensure proper synchronization between producers and consumers using **wait** and **notify**.
5. **Execute the Program:**
   * Run the **ProducerConsumer** class and observe the inter-thread communication.

**Exercise 6: Enhanced Multithreading and Concurrency Features**

**Objectives:**

* Utilize advanced concurrency features in Java.
* Implement thread-safe collections and executors.

**Business Scenario:**

You are developing a web crawler that retrieves and processes web pages concurrently. Use Java's concurrency utilities to manage threads and ensure thread safety.

**Tasks:**

1. **Create a New Java Class:**
   * Create a Java class named **WebCrawler**.
2. **Use Thread Pools:**
   * Use **ExecutorService** to manage a pool of threads for crawling web pages.
   * Implement a thread class named **CrawlerTask** that performs the web page retrieval.
3. **Use Concurrent Collections:**
   * Use thread-safe collections like **ConcurrentHashMap** to store crawled data.
   * Ensure proper synchronization when accessing the collection.
4. **Implement and Test Enhanced Concurrency:**
   * In the **WebCrawler** class, create and start multiple **CrawlerTask** threads using **ExecutorService**.
   * Monitor and log the status of the crawling process.
5. **Execute the Program:**
   * Run the **WebCrawler** class and observe the enhanced multithreading and concurrency features in action.